Security Level:

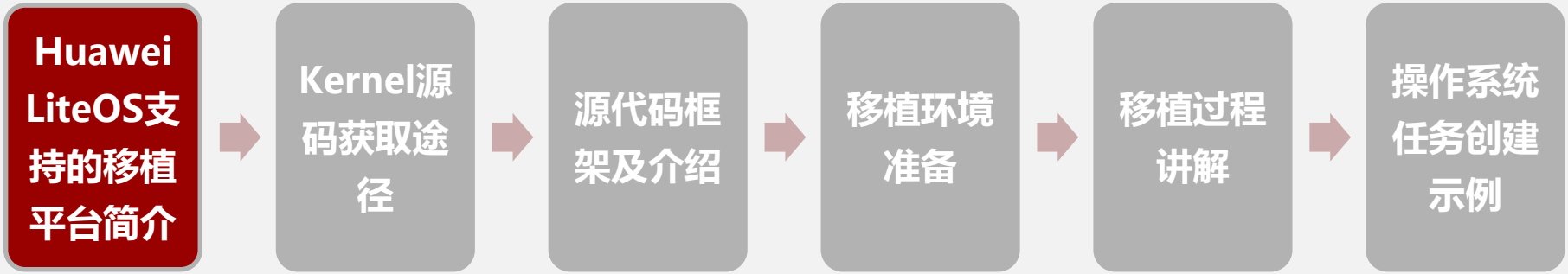
Huawei LiteOS
在STM32系列处理器上的移植

www.huawei.com

HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

提纲


Huawei LiteOS支持的移植平台简介
Kernel源码获取途径
源代码框架及介绍
移植环境准备
移植过程讲解
操作系统任务创建示例

Huawei LiteOS目前支持的移植平台简介

一： ARM系列处理器

ARM9 ARM11

ARM cortex A系列： ARM cortex A7 ARM cortex A53

ARM cortex M系列： M0， M3， M4， M7

典型示例：海思IPC Camera(ARM cortex A7)

STM32系列处理器

NB-IoT芯片(boudica)

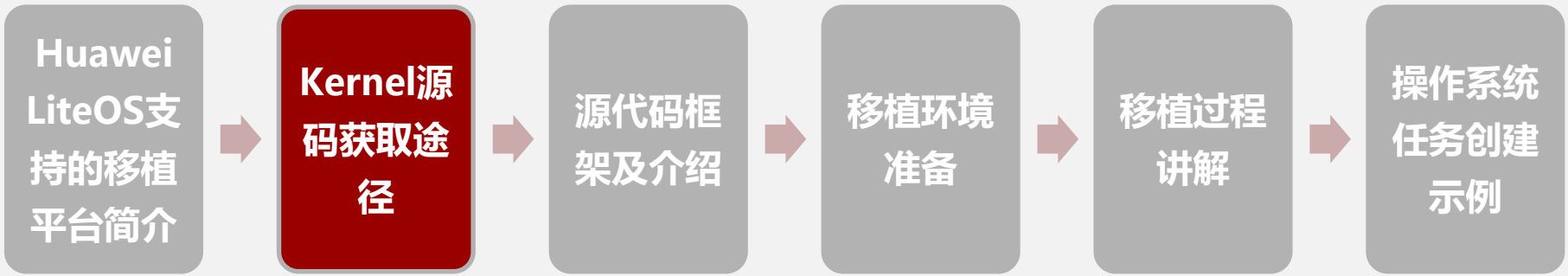
二: Intel 处理器

典型示例： Intel® Quark™ SE平台

三: Tensilica的DSP处理器

典型示例： Xtensa LX7及LX4系列DSP

提纲

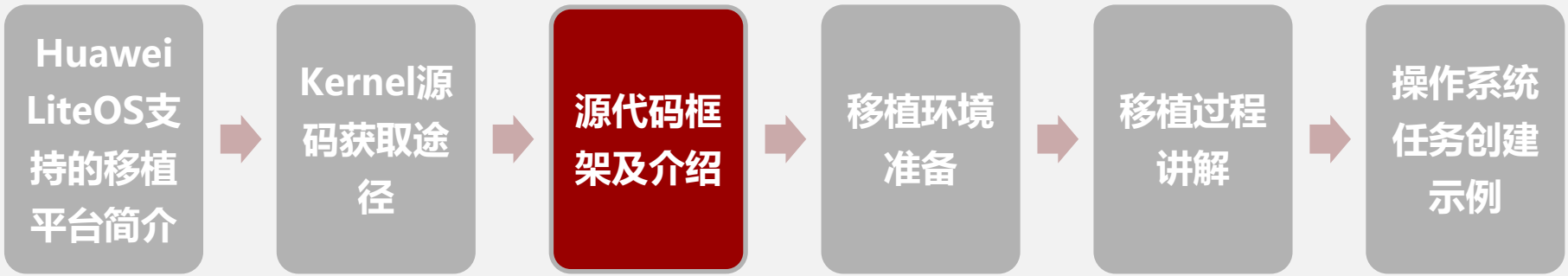

Huawei LiteOS支持的移植平台简介
Kernel源码获取途径
源代码框架及介绍
移植环境准备
移植过程讲解
操作系统任务创建示例

Huawei LiteOS源码获取途径

Huawei LiteOS kernel源码下载地址

华为开发者社区：
http://developer.huawei.com/ict/cn/site-iot/product/liteos
Github：
https://github.com/Huawei/Huawei_LiteOS_Kernel

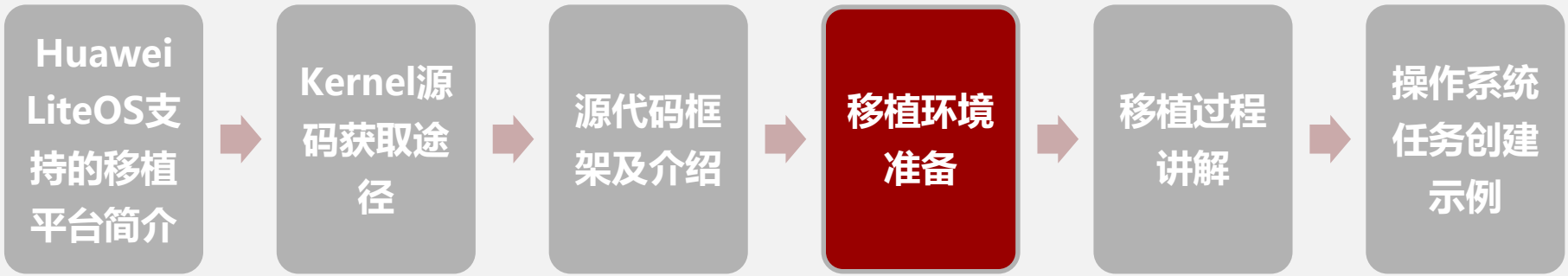
提纲


Huawei LiteOS支持的移植平台简介
Kernel源码获取途径
源代码框架及介绍
移植环境准备
移植过程讲解
操作系统任务创建示例

源代码框架及介绍

序号	一级目录	二级目录	说明
1	Kernel	Base	平台无关的内核代码
		Include	内核的相关头文件存放目录
2	Platform	bsp	系统配置文件 应用入口相关示例代码
		cpu	硬件体系架构相关代码 汇编调度代码

提纲


Huawei LiteOS支持的移植平台简介
Kernel源码获取途径
源代码框架及介绍
移植环境准备
移植过程讲解
操作系统任务创建示例

硬件环境需求

- **开发板硬件：** 板载STM32F103、 STM32F4、 STM32F7 全系列芯片中任意一款开发板或者最小系统板。
- **仿真器：**ULINK、JLINK、ST-Link、符合CMSIS-DAP标准的Debugger等。
- **串口模块：**开发板、最小系统板板载USB转串口模块或者RS232串口，没有的话也可自行外接USB转TTL模块（CP2102 CH340 PL2303等USB转TTL模块）。
- **外设：**GPIO可控的LED指示灯，用来创建Demo应用。

本次移植使用的第三方开发板：
秉火指南者 STM32F103 开发板

MCU型号：STM32F103VET6
RAM：64K
FLASH：512K
MCU主频：72M
板载CH340 USB转串口模块
仿真器：秉火开发板配套的CMSIS-DAP标准的Debugger

软件环境需求

- 主流的 ARM cortex M 系列微控制器集成开发环境

  IAR　　华为开发者社区开源的工程基于该IDE

  GCC + Eclipse　　需要自行安装插件，调试环境需要配置

  MDK　　本次移植使用的IDE

- MDK安装需求

  1.安装MDK5.2.1　　下载地址：

  https://www.keil.com/demo/eval/arm.htm

  2.安装芯片对应pack　下载地址：

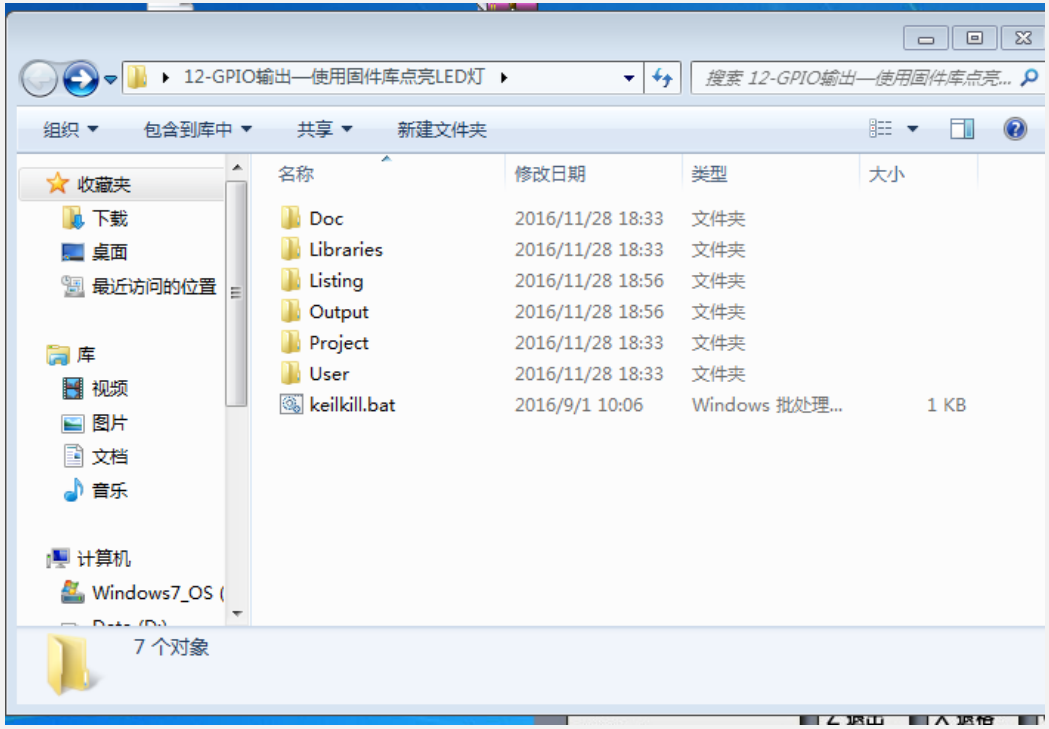
  http://www.keil.com/dd2/Pack/#/eula-container

准备第三方STM32 开发板裸机工程模板

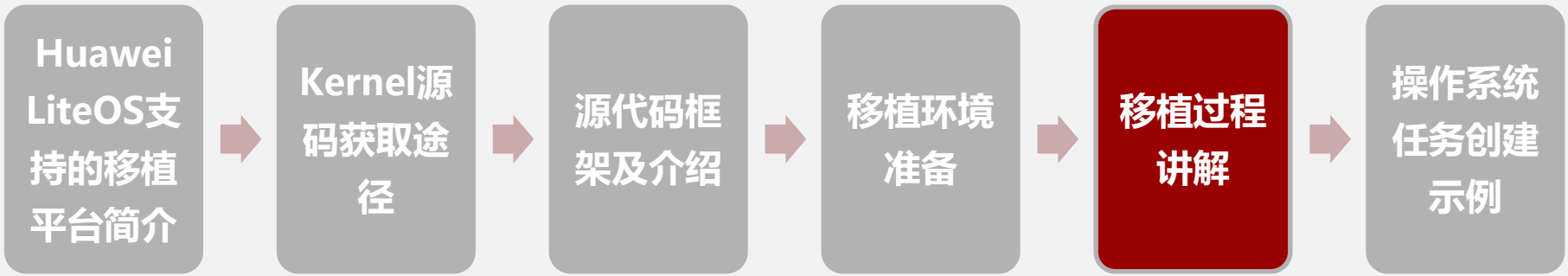
- 获取你需要移植的开发板的资料，包括开发板例程和硬件原理图

我们移植需要的秉火指南者STM32F103开发板资料需要到秉火官方论坛下载，地址： http://www.firebbs.cn/

选取一个STM32裸机工程模板作为Huawei LiteOS移植的基础环境



提纲


Huawei LiteOS支持的移植平台简介
Kernel源码获取途径
源代码框架及介绍
移植环境准备
移植过程讲解
操作系统任务创建示例

移植Huawei LiteOS的主要步骤概述

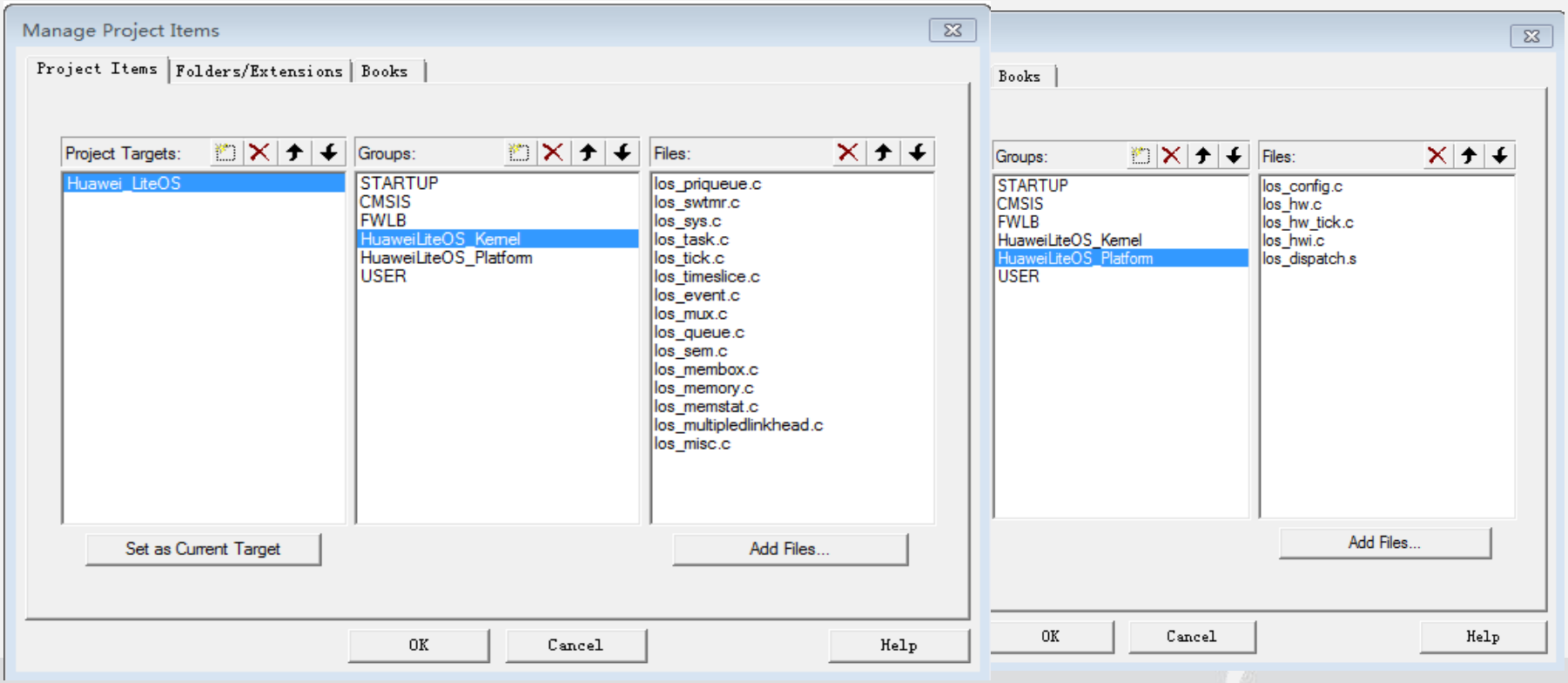
- 1. 在集成开发环境中添加Huawei LiteOS源码
- 2. 适配系统调度汇编文件(los_dispatch.s)
- 3. 根据芯片类型适配硬件资源（los_hw及los_hwi）
- 4. 配置系统参数(los_config.h)
- 5. 修改分散加载文件
- 6. 解决部分常见移植代码编译错误

步骤一：添加Huawei LiteOS源码

- 将Huawei LiteOS Kernel源码文件夹Huawei_LiteOS拷贝到MDK工程源码目录下;
- MDK工程下新建Huawei LiteOS相关Group，存放Huawei LiteOS内核代码；

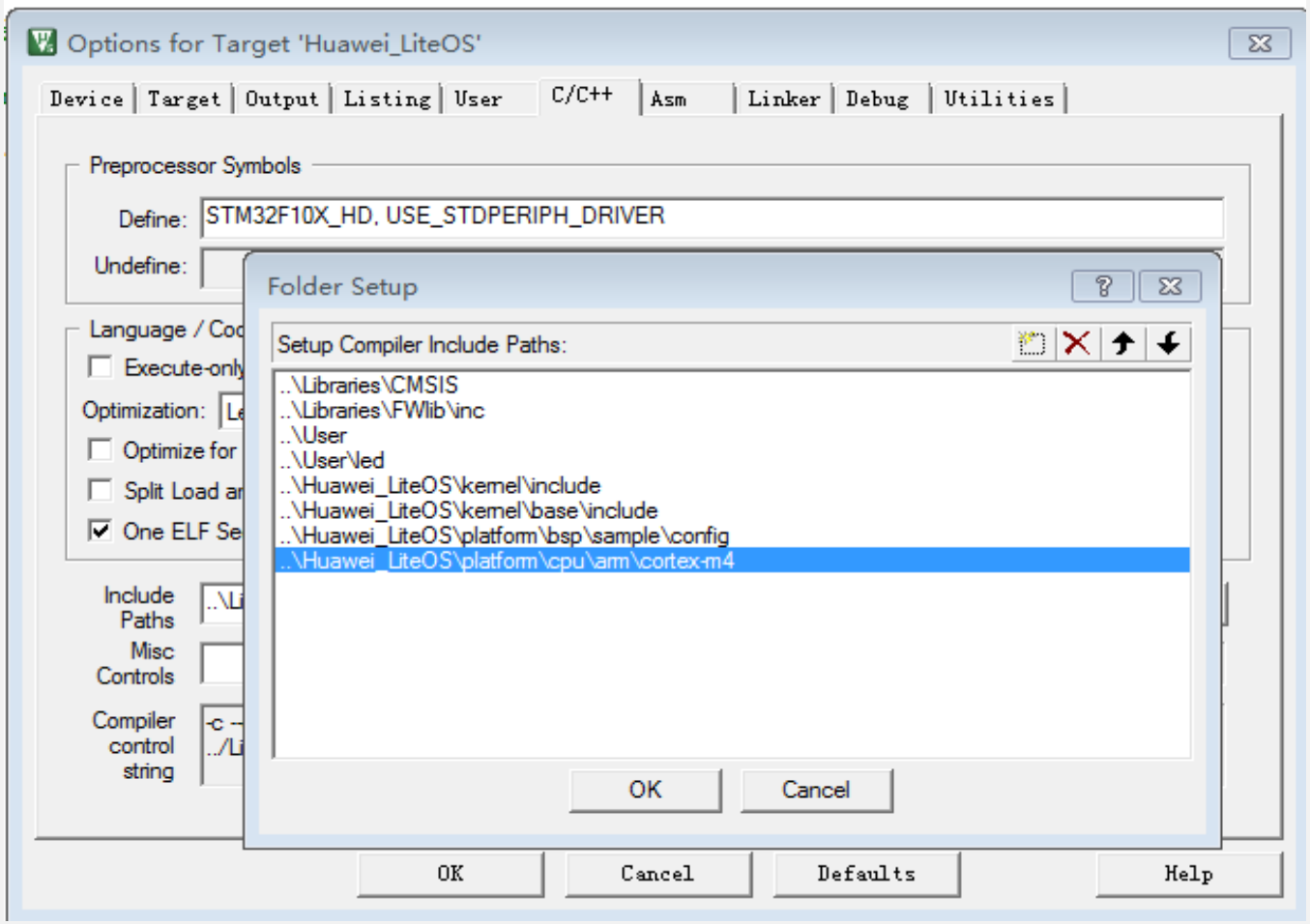
✓ 内核源码位于Huawei_LiteOS\kernel\base目录下，我们把子目录core、ipc、mem、misc目录下的c文件全部添加进来，一共15个文件。

✓ 添加bsp\sample\config下的los_config.c，cpu\arm\cortex-m4子目录下的los_dispatch.s、los_hw.c、los_hw_tick.c、los_hwi.c，一共5个文件。



包含Huawei LiteOS头文件

在MDK工程上右键选择Options for Target...，然后在弹出工程配置对话框中选择C++选项卡，添加包含路径,工程的所有包含目录如下图所示



步骤二 修改系统调度文件los_dispatch.s

- 将los_dispatch.s文件从IAR移植到keil需要修改字段定义关键字

  SECTION .text:CODE(2) 改成 AREA |.text|, CODE, READONLY

- 去除浮点寄存器相关汇编代码，因为我们是从M4移植M3处理器

  删除ADD R12,R12,#72;VPUSH S0；VPOP S0；VSTMDB R0!,{D8-D15};VLDMIA R1!,{D8-15}

- 修改LOS_StartToRun函数

  将ADD R12, R12, #100 改成ADD R12, R12, #36

  添加MSR xPSR, R7 ;写程序状态寄存器

  函数最后加上

  ```
  NOP
  ALIGN ;对齐伪指定
  AREA KERNEL, CODE, READONLY
  THUMB
  ```

- 修改TaskSwitch函数，最后添加

  ```
  NOP
  ALIGN
  END
  ```

步骤三　根据芯片类型适配硬件资源

一：修改los_hw.c和对应的头文件，配置相关寄存器

- 修改los_hw.c文件中的osTaskExit函数（los_hw.c 行号：90左右）

```
LITE_OS_SEC_TEXT_MINOR VOID osTaskExit(VOID)
{
    __disable_irq();
    while(1);
}
```

MDK和IAR中内嵌汇编写法不一致。故修改,也可以使用MDK内嵌汇编

```
__asm VOID osTaskExit(VOID)
    {
                CPSID   I
    }
```

- osTskStackInit 函数中注释掉浮点代码（ los_hw.c 行号：115 ）
- los_hw.h文件中修改TSK_CONTEXT_S结构体，删除浮点相关成员

```
72  typedef struct tagTskContext
73  {
74      UINT32 uwR4;
75      UINT32 uwR5;
76      UINT32 uwR6;
77      UINT32 uwR7;
78      UINT32 uwR8;
79      UINT32 uwR9;
80      UINT32 uwR10;
81      UINT32 uwR11;
82      UINT32 uwPriMask;
83      UINT32 uwR0;
84      UINT32 uwR1;
85      UINT32 uwR2;
86      UINT32 uwR3;
87      UINT32 uwR12;
88      UINT32 uwLR;
89      UINT32 uwPC;
90      UINT32 uwxPSR;
91  } TSK_CONTEXT_S;
92
```

说明：los_hw模块涉及CPU硬件相关配置，移植的时候需要根据具体的CPU资源进行修改，我们移植的是M3，所以删除浮点相关代码，如果是M4或者M7，浮点代码不需要删除。

步骤三　根据芯片类型适配硬件资源

二：修改los_hwi.c和对应的头文件，配置中断

- 根据STM32启动文件修改PendSV_Handler异常向量和SysTick_Handler向量的名称

  Huawei LiteOS源码中，他们分别叫osPendSV、osTickHandler。

  修改办法：

  MDK主界面下使用Ctrl+H将查找到的osPendSV和osTickHandler都分别替换成PendSV_Handler和SysTick_Handler。同时，在STM32F10X_it.c中使用__weak关键字修饰这两个中断服务函数，避免重定义。

- 修改osIntNumGet函数

```
LITE_OS_SEC_TEXT_MINOR __asm UINT32 osIntNumGet(VOID)
{
        MRS R0, IPSR//读取异常 中断状态
        BX LR
}
```

- 在los_hwi.h文件中（行号243）把_BootVectors[]修改成__Vectors[]

步骤四　在los_config.h中配置系统参数

常用参数配置

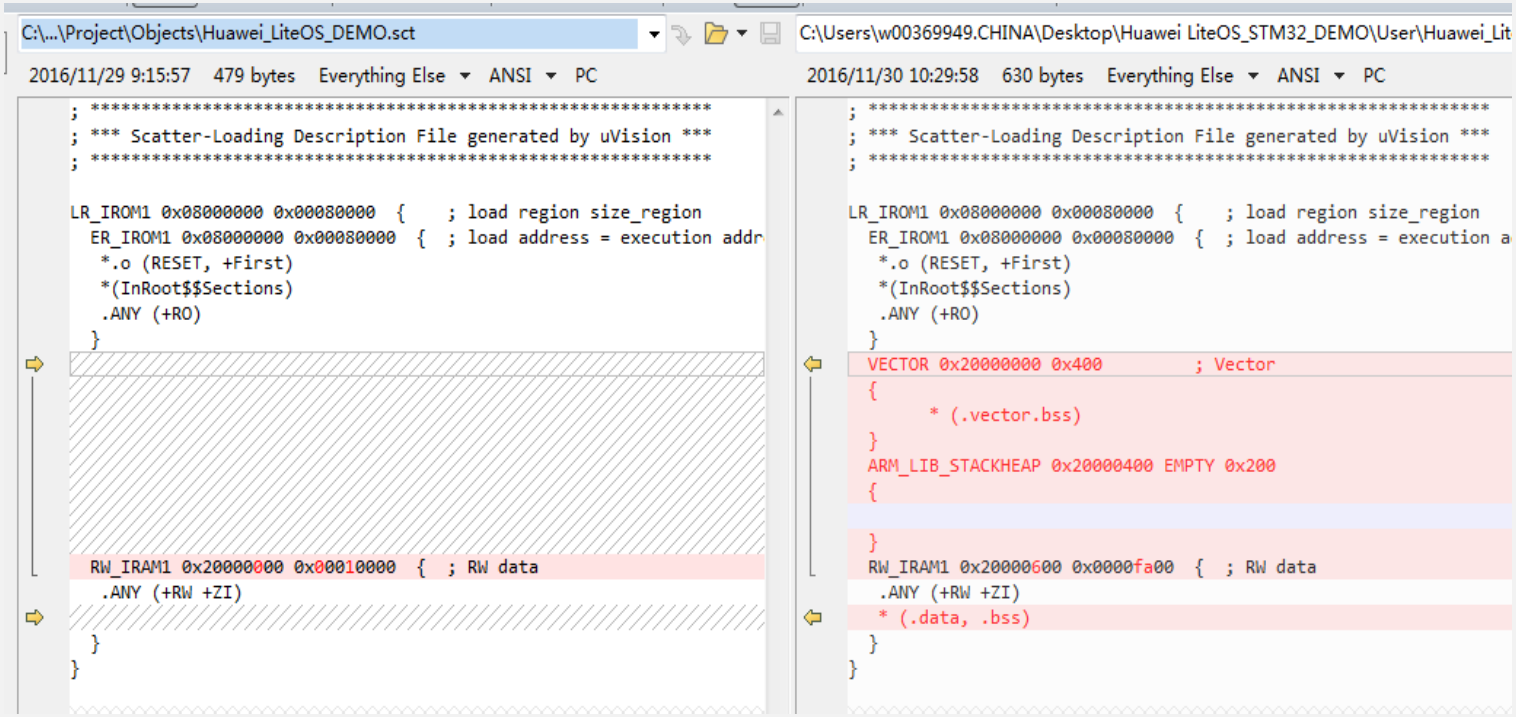
- #define OS_SYS_CLOCK 36000000
- #define LOSCFG_BASE_CORE_TSK_LIMIT 15
- #define OS_SYS_MEM_SIZE 0x00008000
- #define LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE SIZE(0x2D0) // default stack
- #define LOSCFG_BASE_CORE_SWTMR_LIMIT 16
- ....

添加用户任务入口函数

- extern UINT32 osAppInit(VOID);
- 该函数需要用户去实现，用户创建的系统任务都在该函数中注册，该函数会被los_config.c中的系统main函数调用。

步骤五　修改MDK分散加载文件sct

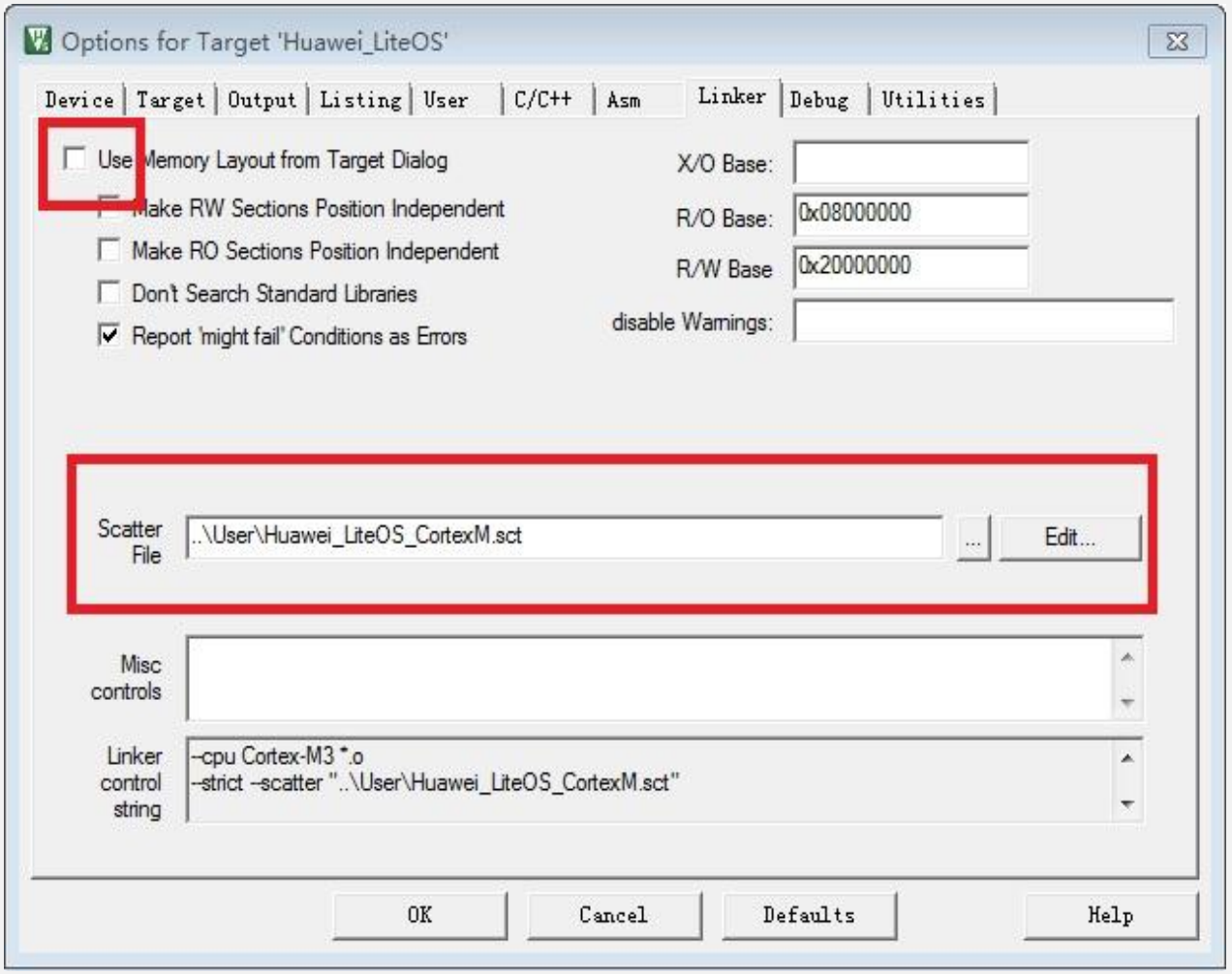
- 由于Huawei LiteOS中对数据和代码位置进行了控制，代码和数据会放在多个不同的内存区域，因此需要使用分散加载文件进行描述，要是系统准确运行起来，需要重新编写一个分散加载文件。



其中.vector.bss需要在los_builddef.h文件中进行配置，将该文件第90行的宏定义注释取消掉，修改后如下：

```
#define LITE_OS_SEC_VEC        __attribute__ ((section(".vector.bss")))
```

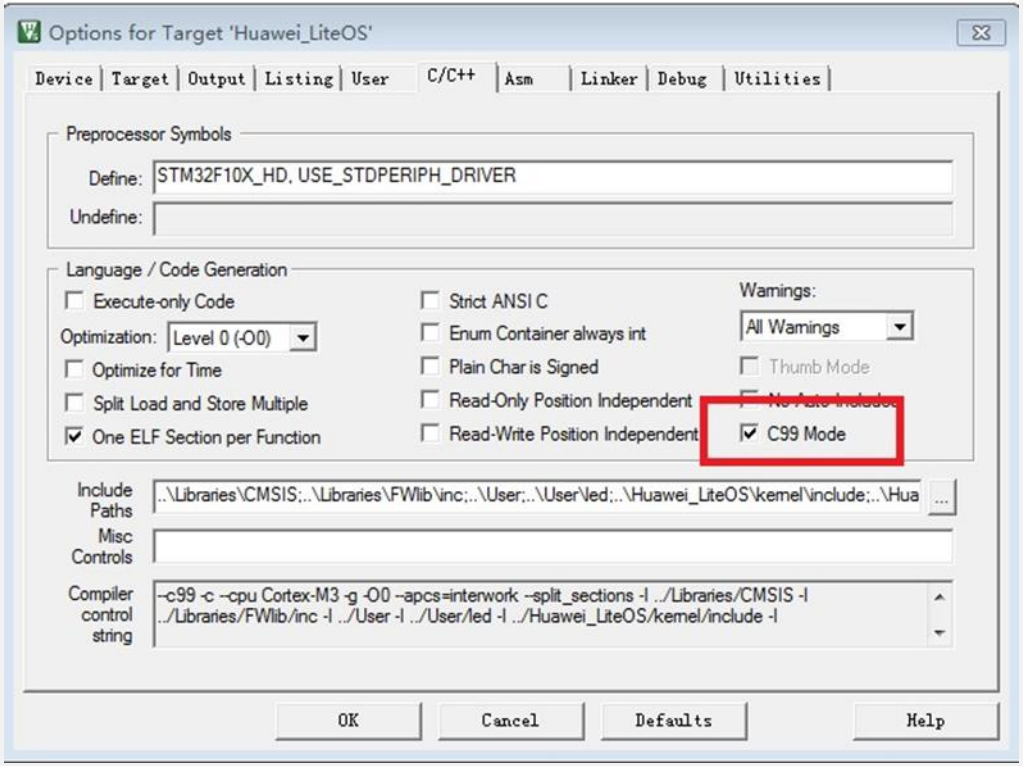
链接分散加载文件


Options for Target 'Huawei_LiteOS'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Errors
X/O Base:
R/O Base:
0x08000000
R/W Base
0x20000000
disable Warnings:
Scatter File
..\User\Huawei_LiteOS_CortexM.sct
...
Edit...
Misc controls
Linker control string
--cpu Cortex-M3 *.o
--strict --scatter "..\User\Huawei_LiteOS_CortexM.sct"
OK
Cancel
Defaults
Help

步骤六 解决部分常见移植代码编译错误

- **错误1：**LITE_OS_SEC_ALW_INLINE INLINE 不识别

  原因static inline 需要C99标准支持，在工程配置中勾选C99 Mode

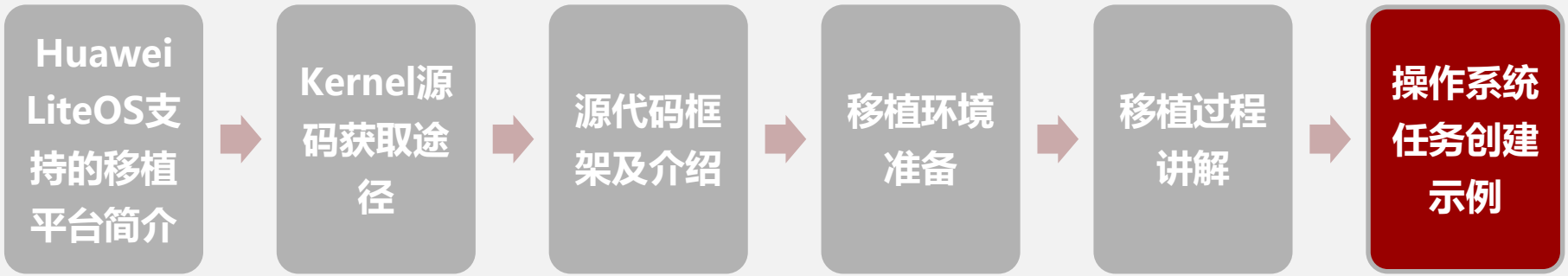


- **错误2：**warning: #161-D: unrecognized #pragma

  注释掉#pragma location = “.vector” ，IAR编译器支持，MDK不支持

IAR中的绝对定位，不是C标准，意思是接下来的数据会存储到.vector段，跟icf文件对应

提纲


Huawei LiteOS支持的移植平台简介
Kernel源码获取途径
源代码框架及介绍
移植环境准备
移植过程讲解
操作系统任务创建示例

Huawei LiteOS任务的创建

- **Huawei LiteOS启动流程**

- **任务创建流程**

1.开发者编写用户任务函数（具体的业务代码）

2.配置任务参数并创建任务

3.实现用户入口函数osAppInit()，注册已经创建的任务，等待操作系统启动后进行任务调度

任务创建示例代码

1.用户任务函数（LED指示灯与串口打印）

```
31  VOID task1(void)
32  {
33    UINT32 uwRet = LOS_OK;
34    UINT32 count=0;
35    while(1)
36    {
37      count++;
38      //printf("this is task 1,count is %d\r\n",count);//串口打印计数值
39      LED1_TOGGLE;//LED指示灯翻转
40      uwRet = LOS_TaskDelay(1000);//操作系统延时函数
41      if(uwRet !=LOS_OK)
42        return;
43    }
44  }
45
```

2.配置任务参数并创建任务

```
46  UINT32 creat_task1(void)
47  {
48    UINT32 uwRet = LOS_OK;
49    TSK_INIT_PARAM_S task_init_param;//任务参数配置结构体
50    task_init_param.usTaskPrio = 0;//任务优先级
51    task_init_param.pcName = "task1";//任务名
52    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task1;//指定入口函数
53    //设置任务堆栈大小
54    task_init_param.uwStackSize = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
55    //设置任务执行完毕后自动删除，默认值，一般不修改
56    task_init_param.uwResved = LOS_TASK_STATUS_DETACHED;
57    //调用create函数创建任务
58    uwRet = LOS_TaskCreate(&g_TestTskHandle,&task_init_param);
59    if(uwRet !=LOS_OK)
60    {
61      return uwRet;
62    }
63    return uwRet;
64  }
65
```

任务创建示例代码

3.实现 osAppInit()函数，注册已创建任务

```
101  UINT32 osAppInit(void)
102  {
103    UINT32 uwRet = 0;
104    hardware_init();//硬件模块初始化
105    uwRet = creat_task1();//添加任务1
106    if(uwRet !=LOS_OK)
107    {
108      return uwRet;
109    }
110
111    uwRet = creat_task2();//添加任务2
112    if(uwRet !=LOS_OK)
113    {
114      return uwRet;
115    }
116    return LOS_OK;
117  }
```

中断创建示例

- 创建一个简单的定时器中断

  1. TIM模块初始化 TIM3_Init
  2. 中断服务程序编写 TIM3_IRQHandler
  3. 调用中断创建接口LOS_HwiCreate创建中断

- 示例代码：

```
//定时器初始化
TIM3_Init(5000-1,9000-1);
//定时器3中断服务函数
void TIM3_IRQHandler(void)
{
   HAL_TIM_IRQHandler(&TIM3_Handler);
}
//回调函数，定时器中断服务函数调用
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
   if(htim==(&TIM3_Handler))
   {
      //LED1=!LED1;      //LED1反转
      printf("\r\n This is a hardware interrupt demo\r\n");
   }
}
LOS_HwiCreate(TIM3_IRQn, 0,0,TIM3_IRQHandler,NULL);
```

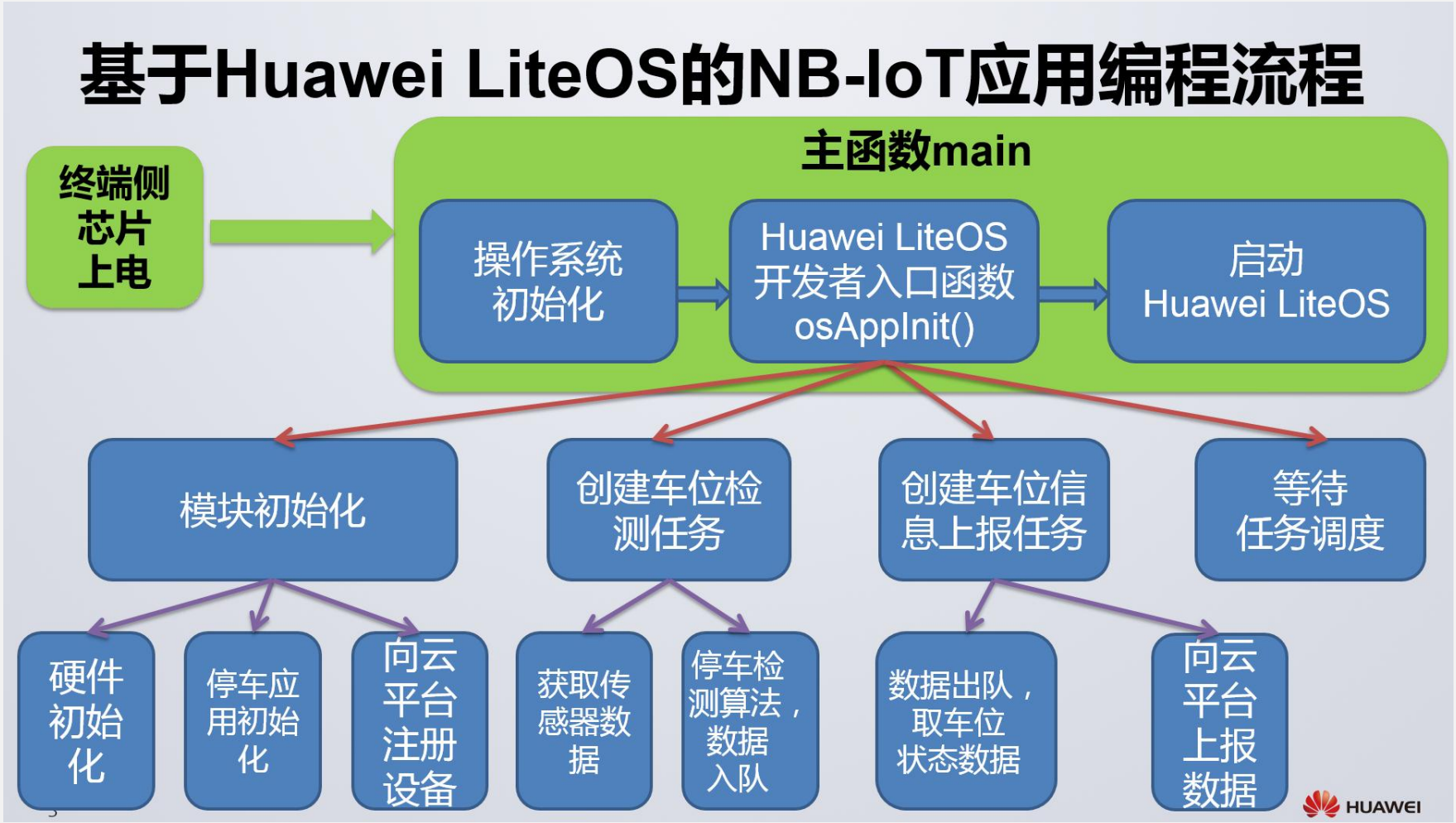
MDK软件仿真配置

//以下代码为软件仿真环境提供Printf函数支持，实际硬件请将Printf函数重定向到串口

```
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port16(n)   (*((volatile unsigned short*)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))
#define DEMCR           (*((volatile unsigned long *)(0xE000EDFC)))
#define TRCENA          0x01000000
struct __FILE { int handle; /* Add whatever needed */ };
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE *f)
{
  if (DEMCR & TRCENA)
  {
    while (ITM_Port32(0) == 0);
    ITM_Port8(0) = ch;
  }
  return(ch);
}
```

//使用软件仿真环境下的Printf函数，请在调试环境下，打开view->Serial windows->Debug (printf) Viewer窗口

典型案例分享


基于Huawei LiteOS的NB-IoT应用编程流程
终端侧芯片上电
主函数main
操作系统初始化
Huawei LiteOS开发者入口函数osAppInit()
启动Huawei LiteOS
模块初始化
创建车位检测任务
创建车位信息上报任务
等待任务调度
硬件初始化
停车应用初始化
向云平台注册设备
获取传感器数据
停车检测算法，数据入队
数据出队，取车位状态数据
向云平台上报数据
HUAWEI

Thank you

www.huawei.com